Drones InSecurity

Manuel Kramer, Martin Schmeißer

Cryptography Research Group,
Institute of Computer Science,
University of Tartu

Research Project in Security

January 2015

Supervisors

Rafik Chaabouni
EPFL (LASEC) & UT

Dr. Amnir Hadachi
UT

Contents

Chapter 1

Introduction

This project is about testing the security of drones and showing the outcome of that. We are two students who are doing their bachelors degree in digital media in the field of computer science. For this semester we made an ERASMUS exchange program and decided to be at the University of Tartu in Estonia. Our home university is the University of Bremen in northern Germany. Firstly we would like to answer one of the most important questions when tackling this topic. What exactly are drones? In general there are three kinds of drones, military, semiprofessional and casual. All three types are air vehicles that are not directly flown by a pilot. They are also called UAV (unmanned aerial vehicle). Military drones are made for everything the military could use them for, combat, spying and more. Semiprofessional drones are the high class drones for normal customers. They are more expensive then casual drones and are of a higher quality in their functionality. For example they are able to lift significantly more weight than their casual counter parts. The last kind is the casual drones, these are the most common drones, for this project we used drones from the company Parrot. They are not as good as the semiprofessional kind, but in terms of security they are the same. Most of the commercial drones do not have any protection. The goals of this project were to test drones and their security. This is divided in two parts. The first part is stream sniffing. This means capturing the video stream that a drone transmits to its connected user and afterwards reconstructing it, to see the actual video footage. All this should be accomplished without being detected, not leaving any traces in the network, that the stream was captured from. The second part of the project is the hijacking of drones. This includes the complete capturing of a targeted drone, with access to all the controls and its video stream. After the capturing, the victim should fly autonomously, following our drone to a location we decide. Furthermore the latter part is again divided by two sections. The first one is ground to air battle, the hijacking is made by a laptop and every processing is done by the laptop. The second section is air to air battle which means

a drone has a small computer added on top of it. These small computers are called Raspberry Pi and should have enough processing power to capture a drone. The drone with the Raspberry Pi will search for drones and capture them, afterwards they will come back and bring the captured drone with them. But that brings us to the next question. Why is it important to hijack a drone? This project is important because it raises the issues of unsecured drones. Most of the commercial drone are not secured. They are controlled over Wi-Fi, but the used network is not protected in any way. That means that everyone with a Wi-Fi supported device can access the drone. Drones can be a security issue in two ways. One is against the owner of the drone, because the drone itself is not protected by any kind of security and therefore can be stolen. This is shown with the hijacking part of our project. That means, that a drone, which is not really cheap, as even the casual drones have a price of about 300.- Eur, is gone forever and the person who bought it has almost no chances in getting it back. Another kind of security issue that comes with those drones is the ability to spy, or to transport small things. In the period of this semester, there were a number of reports of drones spotted in the near vicinity of nuclear power plants [1]. Those were sighted in France and Belgium [2]. Furthermore, another drone was spotted over the French Elyse. The problem that happened and why it was in the news is, that those drones have a high quality camera which can record footage of the seen structures. This could help terrorist to plan potential attacks on those nuclear power plants or reveal classified information to the public. Furthermore the responsible persons of those drone sightings could not be tracked back and are still unknown. The problem that comes with transporting small things is, that those small things could be explosives. The casual drones can fly with 200g to 400g extra weight and that is enough for an aimed attack. To let people know that this is a major problem, a political party in Germany landed a drone directly in front of chancellor Angela Merkel, after it flew above the heads of the crowd. There was no security preventing the drone from landing on the stage [3]. There is a variety of way to use those drones in a harmful way, which luckily did not happen yet. That is why we tried applying these security issues to let the broad public know about this threat. The structure of this report is divided in 3 chapters besides the introduction. The requirements that are needed to fulfil our goal, which means the hardware and the software that is required to reproduce this project. The implementation that describes exactly what we did in this project and how we did it. The stream sniffing, the hijacking and the image processing to track the drone are the main parts of this chapter. The last chapter is the conclusion, where we sum up what happened in this project, what are potential improvements and what problems still exist. Furthermore we state the possible solutions to these problems and discuss what needs to be done.

Chapter 2

Requirements

This section will feature all information needed to rebuild the project. We will cover information about the hardware that was used and how expensive the entire setup was, as well as the software, costs of the used software and problems that were faced while trying to install or use them.

2.1 Hardware

The following table lists the hardware used for this project. It states the name and a description of each used hardware. In total the hardware used had a price of 705.- Eur with both drones included. If one drone is excluded, because only the attacking one is needed, the costs sums up to 405.- Eur.

Name	Description	Price
Parrot AR Drone 2.0	The AR Drone 2.0 is a quad copter drone	300.- Eur
Raspberry Pi B+ Starter kit	Minicomputer with 512 MB RAM, 4 USB 2.0 ports and BCM2835-SoC processor with one core. Includes 8GB micro SD card and USB Wi-Fi adapter	60.- Eur
Alfa AWUS036H USB antenna	Long range USB Wi-Fi adapter (802.11 b/g) used for packet injection with RealTek 8187 chipset	25.- Eur
USB Battery (i-tec Power Bank 2600)	Battery used to power the Raspberry Pi with a USB to Micro-USB connection	20.- Eur

2.1.1 Technical problems with used hardware

While the majority of the hardware worked without any problems, there were still some problems that occurred during the project and this subsection

aims to cover them. Every hardware that had issues will be covered in its on section.

Long Range USB Wi-Fi Adapter

The long range USB Wi-Fi adapter with the model Alfa AWUS036H was used for the packet injection and packet capturing in both parts of the project. The used model is intended for packet injection mainly and therefore caused problems while trying to capture packets. This caused the majority of problems in the first part of the project and it is recommended to take an antenna which is intended for packet sniffing while trying to capture the video stream.

Raspberry Pi B+

The Raspberry Pi B+ caused a range of problems during the project based on its low processing power. The model of the Raspberry Pi series that we used, did not upgrade the used processor and therefore lacks some power that is really needed for image processing, even though it is the newest version to this day. To get the Raspberry Pi working in the intended way there needs to be a lot optimization done to the image processing part of the used code. Even that does not ensure that it will work in the end. Another problem the Raspberry Pi caused, was the usage of software, since it only supports certain versions of certain software. This made setting up the device harder than expected. More on that matter is covered in the software problems.

Drones

The Parrot AR Drone 2.0 is one of the most popular drones and comes with an affordable price. But the drones are not capable of carrying too much weight, which makes it harder to attach the Raspberry Pi, the antenna and the batteries on top of it. This problem can be solved with proper weight distribution or spending extra money on a more expensive drone for the attacking part. More information regarding weights and boundaries of the drones is provided in a section further down.

2.2 Software

The following table lists the software used for the project. It states the name, the used version and a description of the listed software.

Name	Version	Description
Kali	1.0.9a	Linux based operating system installed on the Raspberry Pi
Python	2.7	High-level programming language used for the autonomous flying
OpenCV	2.4.9	Library of programming functions used for image processing
LibARDrone	N.A.	Library of Python functions used for controlling of drones
Numpy	1:1.6.2-1.2	Library of programming functions used for calculations
Drone Control	N.A.	Program written in Python by us for controlling a drone
Shell Scritps	N.A.	Scripts used for finding target drones, hijacking and executing the drone control written by us
TCPFlow	1.4.4+repack1-3	Linux based tool for capturing and reconstructing TCP packets
Airecrack Suite	1.2-rc1-0kali1	Suite with tools for monitor mode, packet sniffing, packet replaying

2.2.1 Pricing of software

All the used software are available for free on the internet. It takes some time to set all of them up with their right version, but this is the only part which hinders the execution of an attack, if the needed hardware is provided.

2.2.2 Additional repositories and libraries

In order to be able to install all the needed software, some additional repositories are needed, which Kali does not provide by default. Even though most of the software needed can be installed using the repositories provided by Kali, many of the versions are outdated and the repositories do not offer newer versions. The additional libraries are listed in the table down below. Aside from the repositories, a number of libraries which are needed for some of the software were also missing and needed to be installed. To enhance the installation progress, we set up a shell script which automatically adds the needed repositories, updates and upgrades those sources and afterwards adds the needed libraries. But while using the script it has to be kept in mind, that those libraries are intended for the Raspberry Pi B+ and the Kali version running on it. If the setup is in any way different, we suggest not to install the libraries and only to add the repositories.

Name	Repository
Jon Severinsson FFMPEG	deb http://ppa.launchpad.net/jon-severinsson/ffmpeg/ubuntu trusty main
Jon Severinsson FFMPEG	deb-src http://ppa.launchpad.net/jon-severinsson/ffmpeg/ubuntu trusty main
Testing - main, contrib and non-free branches	deb http://http.us.debian.org/debian testing main non-free contrib
Testing - main, contrib and non-free branches	deb-src http://http.us.debian.org/debian testing main non-free contrib
Testing - security updates	deb http://security.debian.org/ testing/updates main contrib non-free
Testing - security updates	deb-src http://security.debian.org/ testing/updates main contrib non-free

2.2.3 Problems with software

Setting up the software was connected with problems and many of them will be faced by anyone trying to reproduce the same set up. Therefore this section covers all the problems that occured and the solutions used to fix them.

Operating system

Using an operating system which runs on the Raspberry Pi and supports all the software that was needed for this project was crucial. There were many different systems tested during the setup, but all of them had their fair share of issues. In the end we chose Kali for the project, since it ran on the Raspberry Pi, had many of the software preinstalled and was free of charge. It still has some problems with installing some of the other software requirements, but solutions for them exist.

OpenCV and Kali

The repositories that Kali supports, offer an outdated version of the OpenCV plugin used for the project. Some of the used functions provided by OpenCV work on the outdated version, but some crucial ones do not. The internet provides many different solutions to update the OpenCV version, but most of those solutions end up breaking Kali due to many unsupported libraries. On the Raspberry Pi the issue did not come up, since the previous installed libraries were intended for this device and are the ones required for the newer OpenCV version. So, if installing on a Raspberry Pi B+ running the used Kali version, it is enough to install OpenCV from the additional testing repository added previously. If on the other hand the project is set up on a laptop using a different Linux kernel, there are different steps that are

needed to be taken in order to get OpenCV up and running. Some research pointed us towards a guide [4] explaining how to update OpenCV properly. At first avoid installing the libraries as stated before, and rather verify that the natural dependencies are in place. This can be checked by executing the **aptitude build-dep libopencv-dev** command line in the terminal, as the root of the machine (e.g. using sudo as prefix or logging in as the root). The next thing you want to do, is removing compromised dependencies. You can do so by using these command lines in the terminal. Again make sure that you are in root or using the sudo prefix.
aptitude purge libavcodec-dev libavformat-dev libswscale-dev
aptitude purge libx264-dev x264

When this is done, the new dependencies can be installed to the machine. This is achieved by using the following lines as root.

aptitude install autoconf automake build-essential libass-dev libfreetype6-dev libgpac-dev libsdl1.2-dev libtheora-dev libtool libva-dev libvdpau-dev libvorbis-dev libx11-dev libxext-dev libxfixes-dev pkg-config texi2html zlib1g-dev
aptitude install yasm

Now that all the dependencies are present, we can start to install the H264 codec, FFMPEG and lastly OpenCV. The H264 codec and FFMPEG are needed for OpenCV to work properly. At first we focus on the H264 codec, which is installed by using the following commands. The first part of the installation does not require root access even though having root access will cause no problems.

mkdir ~/ffmpeg_sources
cd ~/ffmpeg_sources
wget http://download.videolan.org/pub/x264/snapshots/last_x264.tar.bz2
tar xjvf last_x264.tar.bz2
cd x264-snapshot*
./configure –enable-shared –enable-pic
make

Those lines will download the package, which is to be installed, extracts it and sets everything up for installation. This process might take a while depending on the internet connection and machine you are running it on. Once everything is done and you are again able to insert commands into the terminal, finish off the installation by using the next two lines. But this time make sure that you are performing the action as root.

make install

ldconfig v

Now that the codec is installed, we can continue with installing FFMPEG. The procedure is quite similar because we are again downloading the package and preparing it for installation. But the configuration for the installation is a little bit different this time around. The first part can again be performed as a regular user and does not need root rights.

cd ∼/ffmpeg_sources
wget http://ffmpeg.org/releases/ffmpeg-snapshot.tar.bz2
tar xjvf ffmpeg-snapshot.tar.bz2
cd ffmpeg
./configure –extra-libs="-ldl" –enable-gpl –enable-libass –enable-libfreetype –enable-libtheora –enable-libvorbis –enable-libx264 –enable-nonfree –enable-x11grab –enable-shared –enable-pic
make

Again we finish the installation with the following commands executed as root.

make install
ldconfig v

Now with the right codec and FFMPEG in place, we can go ahead and install OpenCV with a similar procedure. We download the package, unpack it and install it using Cmake. As with the first two installations, the first part can again be performed as a regular user while the last part needs root rights.

mkdir ∼/opencv_sources
cd ∼/opencv_sources
wget http://kent.dl.sourceforge.net/project/opencvlibrary/opencv-unix/2.4.9/opencv-2.4.9.zip
unzip opencv-2.4.9.zip
cd opencv-2.4.9
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_NEW_PYTHON_SUPPORT=ON ..
make

followed by these commands as root:

make install

ldconfig -v

After all these steps are done, OpenCV should be installed and working. To check, if the right version is installed and that everything works properly, you can use the following technique. Open the terminal and type:

python

This will then start python within the terminal. Now that this is done, you can proceed to see if importing OpenCV works. This can be done by using the following command.

import cv2

If there were no error messages yet then OpenCV is installed and can be imported into any Python project from now on. If you want to check the version, OpenCV is running on, you can check it by using this last command.

cv2.__version__

This command displays the current version of OpenCV, which has been imported to Python. If everything worked, it should now display the version 2.4.9. To exit Python within the console, either close the terminal or use this command:

exit()

Now everything related to OpenCV is set up and it is ready to be used.

TCPflow update

To be able to use TCPflow in the way that it was intended to, an update is required since the version shipped with Kali is far outdated. The version that Kali supports is 0.21.ds1-7 0, but the one needed is 1.4.4+repack1-3. This one can be installed from the testing repository, which has been added manually.

2.3 Layout

The heart of the project's setup is based around a Raspberry Pi B+, which is attached on top of a Parrot AR Drone 2.0. The Raspberry Pi itself is powered by an external battery and is using Kali as the operating system. In addition to that, it has two devices plugged into its USB ports. The first of the two devices is a USB Wi-Fi adapter for basic Wi-Fi coverage, while

the second one is an AWUS036H USB antenna used for packet injection. It may be necessary to plug the second antenna in two USB ports, to get enough power. For storage the Raspberry Pi uses a micro SD card with 8 Gigabytes of space. On this SD card the needed software and files mentioned in software requirements are stored. The drone, which the Raspberry Pi is attached to, has a specific sticker applied to its back side, which is used for the detection in the autonomous flying. The entire setup can be seen in figure 2.1. As a refrence, figure 2.2 shows how the described layout plan could be applied to an actual drone.

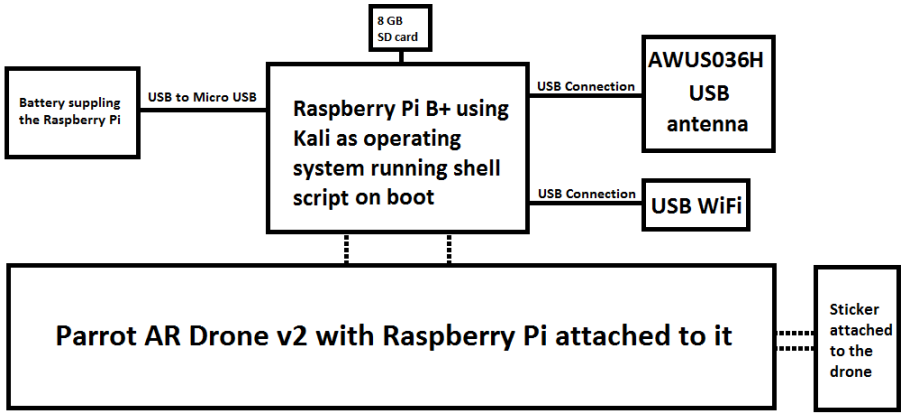

Figure 2.1: Layout plan of the setup

2.4 Specifications

This section will cover the specifications of weight, transmission range and flight time acquired by testing.

2.4.1 Summary Of Results

The weight was measured from the drone itself and the parts that were attached to it. The drone core and one hull, external or internal, determine the basic weight of a flying drone. Furthermore the battery for the drone needs to be included as well, different battery types may change the weight. The first addition is the Raspberry Pi. It consists of the device, the USB battery for its power supply and the USB cable to connect the two. The last addition are the wireless antennas. One for the hijacking and one for connecting. This consists of the main body, the antenna itself and the USB

Figure 2.2: A reference of how the layout could look like with an actual drone

cable to connect it to the Raspberry Pi. All this sums up to 711g for outdoor use. Further installations are described in the table below. The maximum range, where the stream is still running goes up to 105m, that means that the captured drone could still be controlled with the tracking algorithm up to this distance. The drone can fly permanently for 15 minutes and 2 seconds with a HD battery.

2.4.2 Measurements

The table below shows the measured weights of all components and the sums of different combinations.

Nr.	Name	Weight
1	Drone Core	294g
2	Indoor Hull	59g
3	Outdoor Hull	29g
4	Wireless Antenna (large antenna)	99g
5	Wireless Antenna (small antenna)	69g
6	HD Battery (1500mAh)	118g
7	Raspberry Battery	77g
8	Raspberry Pi + Hull	78g
9	Raspberry Pi	44g
10	Raspberry Usb cable	25g
11	Antenna Usb cable	37g
12	Normal Use Outdoor (1,3,6)	441g
13	Normal Use Indoor (1,2,6)	471g
14	Extra Setup (4,7,8,10,11)	316g
15	Extra Setup Tweak (4,7,9,10,11)	282g
16	Extra Setup Tweak 2 (5,7,9,10,11)	252g
17	Full Weight (12,15)	723g
18	Full Weight 2 (12,16)	693g

Chapter 3

Implementation

The implementation of the project will be split into three large sections, covering everything we have done in order to get a working prototype. Each of the sections will have a description of what we did, as well as a subsection on problems we faced, fixed, had to work around, or still exist in the final prototype. The sections in question are the stream sniffing, the hijacking of drones and the autopilot used in combination with hijacking. The first section will explain the stream sniffing part of the project. Then, the hijacking of drones will be explained. Last but not least, the auto pilot used in combination with hijacking will be covered.

3.1 Stream Sniffing

The stream sniffing part of the project was intended to eavesdrop a video stream transmitted by a drone, without leaving evidence of doing so. We first state our approach, with our attempts and the final prototype implementation. We explain in a second part the problems that we faced during the implementation and potential fixes for them.

3.1.1 Approach

In order to sniff the video stream, we needed two things. To start, we needed a software capable of capturing the transmitted stream and also we needed another software capable of showing the captured stream. There are two programs that come to mind. For capturing the packets the program TCPdump could be used, since it is one of the more popular tools for network and packet sniffing. To be able to show the stream, we went for FFMPEG and its tool FFPLAY, which allows the playback and streaming of videos. Streaming the video stream directly from the drone works flawless with FFPLAY. This can be done by typing "**ffplay tcp://192.168.1.1:5555**" into the terminal. Hence FFPLAY is capable of showing the stream itself and

therefore should also be able to show the captured stream. For capturing the packets, we set up a filter with TCPdump, which filtered the MAC address of the drone, that we used for testing. Furthermore we specified the port to sniff on to be 5555, which is the port used by the drone for the video stream transmission. This filter was achieved by using the following command line:

tcpdump -i mon0 '((port 5555) and (ether src host 90:03:B7:EA:8C:C6))' -w *filename*

The **i** option specifies the interface to sniff on. For this example we used mon0, which is an interface put into monitor mode by using **airmon-ng start *interfacename*** . The **w** option is responsible for saving the output as a binary file with a specified name. We used the MAC address of the drone that we were caputring the packets from and therefore it needs to be changed to the address of the used drone when reproduced. Using FFPLAY to play back the created file caused too many problems. After some testing, we changed the **w** option to use - as a name. This directly displays the binary output into the terminal, rather than saving it to a file. Afterwards we used that output to pipe it directly to FFPLAY, which resulted in the following command line:

tcpdump -i wlan1 '((port 5555) and (ether src host 90:03:B7:EA:8C:C6))' -w - | ffplay pipe:

The problem with FFPLAY was still present at this point, but we found a way to directly send the captured packets to FFPLAY without having to save them first. We also managed to figure out where the problem lies, while looking at the manual page of TCPdump. The program captures TCP packets and outputs the entire packet including the header of each packet. FFPLAY is capable of decoding a TCP stream by removing the header and sorting the packets in a way that it recreates the transmitted frame. But this technique is only used when specifying a TCP source, which cannot be a file or piped input. That means we have to remove the TCP header and reorder the captured packets before piping them to FFPLAY. Luckily the program TCPflow captures packets the same way as TCPdump does it, but in addition removes the header and reorders them. It also only returns an output, if an entire group of packets has been gathered. Using the output of TCPflow by piping it to FFPLAY gave inconsistent results. When starting the capturing before the target started the stream transmission, a captured stream was shown by FFPLAY, but starting the capturing after the transmission started, caused FFPLAY to be unable to recreate the stream. After minor research we found the reason for this behavior to be an unknown video codec. The codec is the first thing that gets transmitted upon initialization of the stream. Therefore we were able to recreate the stream when we cap-

tured the first packets, but unable when jumping in, in the middle of the stream. This problem had the simple solution of specifying the codec to be used within FFPLAY. That resulted in the following and final command line:

tcpflow -i mon0 '((port 5555) and (ether src host 90:03:B7:CD:3A:39))' -CB | ffplay -f h264 pipe:

The **CB** modifier specifies that the output should drop the TCP header and then convert it to binary to be able to pipe it towards FFPLAY.

3.1.2 Problem Statement

During the implementation of the stream sniffing, we faced multiple problems regarding the video stream. The major issue was the lack of packets captured by our antenna, which was shown when using the previously mentioned TCPdump. When the device we were trying to capture the stream with was the one connected to the video stream, the amount of packets captured increased significantly, but it still lost a consequent number of packets. While connected, we were able to show a corrupted version of the video stream, where one was able to guess elements from the actual image. However, when another device is connected to the video stream, too few packets are captured to recreate even one frame of the stream. Also rearranging the position of the antenna to a spot where it was located in between the drone and the streaming device improved the amount of packets captured. Due to our testing and the knowledge that the used antenna is mainly used for packet injection and better antennas for packet capturing exist, we are certain that a different antenna could have improved the project. Unfortunately we were unable to perform further tests.

3.2 Hijacking of Drones

The hijacking of drones consists of several shell scripts, which are working hand in hand in order to find target drones within Wi-FI range, selecting the closest target, kicking the original user out, setting up a firewall and launching the autopilot described in the next section. The approach will describe how all scripts are working independently and how they are working together to achieve the final result. At the end we state problems without solutions and discuss potential improvements.

3.2.1 Approach

In an effort to structure the approach, this section is split up into subsections for every shell script that was created for the project. Every function and how it works will be described within its corresponding section.

Turning Monitor Mode On

The first shell script, simply called "'monitorOn.sh"', is responsible for turning on monitor mode on a specified interface. In order to use the script, it requires the interface used for the monitor mode as an input, when using the command. The script uses the Airecrack suite and its Airmon component. The first step the script performs, is to turn off a monitor, if one exists, by simply stopping it with the **airmon-ng stop** function. Afterwards the provided check kill function gets called to remove every process that might interfere with the monitor. Subsequently the monitor gets started with the corresponding command **airmon-ng start** ***interfacename***. Lastly the check kill function is used again, to ensure that no process, which could cause problems has restarted.

Sniffing The Network

The following script is split up into two parts. As they are similar they are therefore explained at the same time. The scripts in question are "'find.sh"' and "'findChannel.sh"'. Both scripts are using the monitor mode, which was created using the "'monitorOn.sh"' script previously explained, to sniff the network for drones. This can be done because all drones manufactured by Parrot share simliar sets of MAC addresses, which are used to identify network devices. Each set uses the same starting characters of the address which in our caste are 90:03:B7. Hence the script could be improved to include the few additional sets as well. With this provided, we were able to use Airodump, another component provided by the Airecrack suite, and filter the network by masking the MAC address. The output from this sniffing gets saved into a file for future analyses. The visual terminal output gets dumped into **/dev/null**, which is a location that directly withdraws the input and therefore erases it. Unfortunately Airodump is lacking a function to make it automatically stop after a period of time. This is why it is used in its own script. After calling the function, the script saves its own process id and sleeps for five seconds. These five seconds are used to gather as much information as possible and could be changed, if needed. After the five seconds have passed, the script terminates its own process and closes Airodump this way. This basic setup is used in both variations of the script. The only part that differs, is that "'findChannel.sh"' has an additional input which specifies a channel to sniff on. "'Find.sh"' performs its sniffing on all available channels and is therefore used as a way to find drones in the network. Sniffing on a specific channel is used to set the current channel, the monitor is running on. Different approaches were used to achieve this switch of the channel, but only this solution worked consistently.

Setting Up The Firewall

In an effort to refuse the connection from any network device but the one that is used to connect to the drone, after every user was kicked out, we used a script called "'setupFirewall.sh"', capable of doing this. Unlike most of the other scripts, it was not based on Bash, but used Expect instead. This was necessary, because to do its task, the script has to use Tellnet to dial into the drone. This is only possible because of missing network protection from the drone. The internal Linux based software of the drone supports a tool called Iptables. This tool can be used to create a whitelist of users, that are allowed to access the drone. The script gets a MAC address as input which is then used to call Iptables to refuse all connections from devices with a MAC address unequal to the one the script received as input. The firewall is very weak and can be bypassed by the spoofing of a MAC address. This means that the users spoofs the machines address to any MAC address he desieres, but because of the short timespan of the attack, the weak firewall is enough.

Releasing The Firewall

To regain access to the drone with every network device, another script with the name "'resetFirewall.sh"' was used. This script uses the same technique as "'setupFirewall.sh"', by dialing into the drone using Telnet and then using Iptables to release the firewall. What it does is basically setting everything within the Iptables to accept every connection. This script however can only be used by the machine with the network device, which is allowed to connect to the drone or, if MAC spoofing is used.

Hijacking The Drone

Now that all the important scripts have been listed and described, we will continue with the hijacking script. This is the longest script and also the one responsible for the majority of the attack. It combines many of the previously listed scripts in order to achieve the goal. The code itself is split into various parts with different functions. All parts will be explained and differentiated in their corresponding section to make the structure easier to tell.

Main Loop

The main loop is the starting point for the script. It is responsible for the main initialization and calling the different parts of the script. It starts with going into a loop, which only breaks once a valid target was found. The loop starts by deleting a sniffed file, if one still exists from previous attacks. Following the deletion, the script calls the "'find.sh"' script in order to create a new output file for the current runtime. It then calls the function, which is responsible for analyzing the output of the sniffing. After the analysis

is done, the loop tries to sort the found drones by calling the appropriate function. After these steps, it will check if a drone was found and will either loop back to the start, if no valid target was found otherwise it will proceed to call the hijacking function.

Analyzing The Output

In an effort to analyze the output of the sniffing provided by the search.sh script, the script goes through each line of the output file. The information that we seek is located in the second part of the output, divided by a unique header line. With every parsed line we check, if it is the unique header line and in the case that it indeed is, the script switches to the analyze mode by setting a boolean for that information to true, which then proceeds to extract information of every following line, until it reaches the end of the file. The analysis at first checks every line, if it includes the starting of the MAC addresses from drones. This method has been mentioned in the "'Sniffing The Network"' section above. The line only gets further analyzed, if a given MAC address is found within it. At this point, it will check, if the line contains a specifically set MAC address which is supposed to be excluded. The reason for this is, if you perform the attack from a drone, your own drone should be excluded from it while targeting the other drones. If the specified address is not found within the current line, the script acknowledges it as a potential target. Now all requirements are checked and the given line can be split in sections. This is possible because all the available information is split by commas, which list the MAC address of the client, the MAC address of the station (drone) and the drone's power. There is some additional information, which is not needed for our analysis and therefore gets discarded. The next step this function does, is to set up an array containing all found drones. Every time a new drone is found, the drone gets added to the array, including its MAC address, the power level and the MAC address of the client connected to the drone. This information is divided by a semicolon. However, if an existing drone is found again with a different client, this client gets attached to the drone's entry in the array. After the analysis reaches the end of the file, the created table containing one entry for each drone, with the information of its MAC, the power level and all MAC addresses of connected users is done. This table will be used for sorting and finding the best drone.

Finding The Best Drone

After the analyses of existing drones within range is done, it is time to find the drone which is the closest and therefore the best target available. All information that is needed to achieve this, has been provided by the analyses of the sniffing output. It uses the table which was created during the analyses and goes through it until all entries are checked. A placeholder power level with the amount of -100 has been set up in order to compare every

drone with it. The value -100 will never be reached within the ranges of power levels and therefore will be replaced by any entry. Every power level of every drone within the provided array gets compared to the current power level, which as stated before starts with -100. Every drone that is closer to the used antenna should have a higher power level. If the current drone's power level is in fact higher than the currently saved one, all information of the current drone, including the power level gets saved as the temporary best drone. This procedure repeats itself until every drone has been checked. After the table has been processed, the drone currently saved as best result will be taken as the target for the attack. With this, the script extracts the channel on which the found drone is broadcasting and the ESSID of the drone. This is achieved by filtering the exact MAC address and by extracting the information from the corresponding part of the found line. All this information is saved within final variables which refer to the definitive target.

Executing the attack
At this stage the gathering of information is done and the attack can be performed. The script resets the channel on which the monitor is searching, by restarting the monitor using the "'monitorOn.sh"' script. This will disable the monitor, kill all interfering processes and start the monitor again. To fix the channel, the script "'searchChannel.sh"', which is used for sniffing on a specific channel, gets launched with the previously found channel. Now that the attack is all set up, the code goes through each detected client that is connected to the drone and de-authenticates them by replaying 10 deauthentication messages to each of them. After every user has been kicked out of the drone, the interface which has been specified at the launch of the script gets re-enabled and connects to the target drone. When the final connection has been established, the script extracts the MAC address of the interface used for the connection from a file provided by the operating system. With this MAC address the script for setting up the firewall gets launched. This way no device except the interface that the script used for the connection can connect to the drone from now on. In order to finish the attack, the Python program for auto piloting, which is described in section 3.3, gets executed.

Combining The Scripts

The last step that has been taken by us, was to create a small combining script, for setting up the monitor and executing the attack. The script in question is the "'bootingFindTarget.sh"' and it calls the necessary scripts with the input parameters used on our system. This way the entire attack can easily be put into the auto start of the system.

3.2.2 Problem Statement

Even though the final prototype of the drone hijacking is working without problems, final tweaks could be brought. The way that the output of the sniffing is achieved can be improved and the entire procedure can be cleaned up. At this point, the way that the script is extracting information from the output is way too complex. Changing it to reduce the amount of work done for parsing would already be an improvement. In addition, sorting drones could be done while parsing the script. For example by dropping drones if a better one was already found or replacing older ones when a better one is found. This again would minimize the required work needed for the drone detection. Another thing that could be tweaked is the timing of the network sniffing. Right now it takes five seconds to sniff the network for drones and another five seconds to switch the channel to the one that the found drone is using. Further testing could show an improved timing, which could be used for sniffing. Also an alternate way of switching the channel could be used, if one is found that does so in a consistent way. Another improvement that could be made, is to include further MAC addresses to the list of drone MAC addresses. As stated in the approach, currently only drones manufactured by Parrot are detected. Other manufactures also have fixed sets of MAC addresses for all their drones, which then could be added to the scripts.
To conclude, there is still room for tweaks and improvements, but the given prototype is working consistently and does not cause performance issues up to this point.

3.3 Autopilot

The autopilot in this project is written in Python using the OpenCV library. It is using the stream of the captured drone to determine the location of our drone and follow it to a destination we choose, if nothing is found the drone is rotating to search for the drone. The stream, using image processing, is altered to track the sticker of our drone. The sticker is made of three rectangles filled with the colors orange and blue. The autonomous flying is using the LibARdrone library to apply controls over the laptop or the Raspberry Pi.

3.3.1 Approach

The approach that is used in the running prototype is a relatively simple color detection. That means, we are using the colored sticker on our drone. The stream gets captured by the laptop and will be divided into frames. Each is going to be processed, with the color detection we wrote. The first step to make the color detection is to define the colors that are going to be tracked. The sticker, like mentioned above, is made of two colors and

those are the two color ranges that were implemented. Those ranges are represented as arrays and will be applied to the frame. The result is a mask of the image from the drone. A mask is an image that is made of black and white colors. For this algorithm every pixel that is within the color ranges is white, everything else is black. Both masks that we got from the two color ranges will be joined into one image for further processing. To get a smoother white area in the image and less black pixels in the white blob area, we use the morphologyEx function in the OpenCV library. This function will check surrounding pixels and fill them with the majority of those. At the end, we hopefully have one white area, instead of two or three areas. The detection part is done by finding the contours of this image. This function tries to detect the shape of everything in an image. Because the contours would find many shapes in the image we made the masks in the beginning, to make it as simple as possible so that the program will find exactly what we are looking for. The contours will give the coordinates of the shape. With those coordinates we draw a rectangle around the found white area. Even if the color range is well done, there will be false detections which result in areas marked that should not be detected at all. The next goal is to search for the best detection. To accomplish this, we compare the histogram of the content of the rectangles with a template that was made beforehand. This template is a picture that only consists of the sticker without different colors mixing in. The histogram we make, will show the distribution of the RGB Values in the whole picture within one graph. This histogram is then compared with the histograms of the detected areas and the best match is used as the result. With this method, only one rectangle is used in the further process. To see the detection, the rectangle is drawn into the frame as shown in figure 3.1.

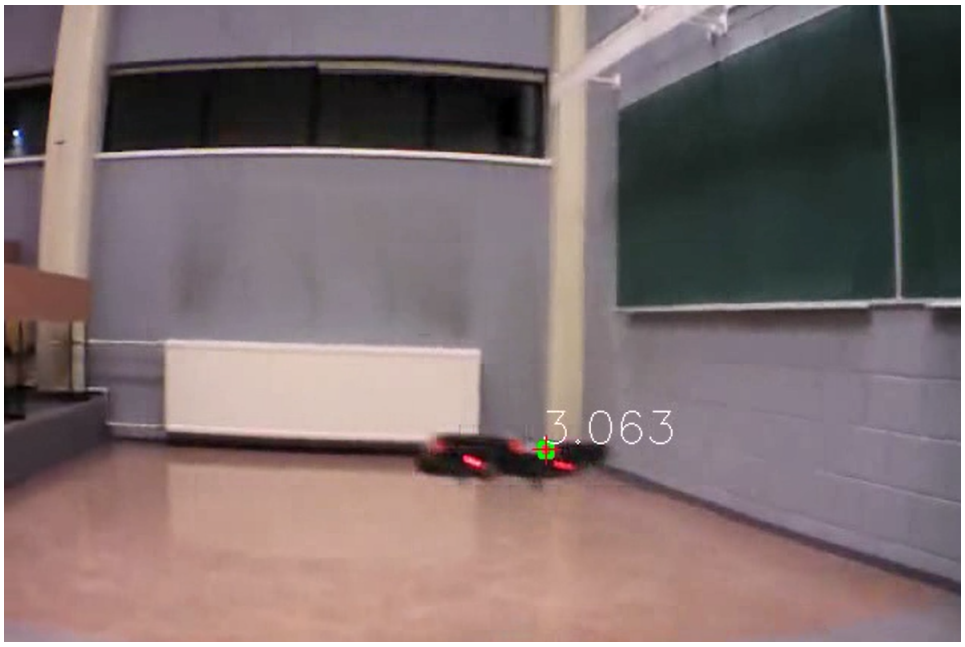

Figure 3.1: The output image created from the processed stream

Now that the system can detect the sticker, it needs to adjust the controls of the drone to let it fly automatically. In order to choose when the drone needs to get which signal, the frame is divided into five sections. Those five sections are the top and bottom, the left and right and the centre of the frame. The middle of the rectangle is used to determine the location of the drone in the frame. We prioritised the up and down flying and only afterwards adjust the rotation of the drone, until the drone is centred. Then the drone will fly forward towards the sticker on the drone. In order to not let the drone crash into the other drone, a distance measurement algorithm is implemented that takes into account, the height of the detection in the image, the actual height of the sticker, the focal length and sensor height of the camera and the height of the whole image.

3.3.2 Problem Statement

The tracking part of the drone was the hardest part. We tried several solutions to make a correct detection. In the beginning we tried to use machine learning. This means to let the computer learn how a drone looks like. To accomplish this we used a machine learning algorithm, that is called haar cascade. In order to get a good result, we needed many pictures with a drone and without a drone in it. Afterwards the algorithm would compare all these pictures with each other to have a concept of the drone, that we could track. This kind of approach needs a lot of processing power and we

had to drop this kind of tracking, because the machine learning would not finish due to limitation of our resources. Afterwards we tried to implement a detection based on the SURF or SIFT algorithm provided in the OpenCV library. Both are feature detecting algorithms, which are just to find and track given features of an image. SURF is a newer algorithm and claims to be faster than SIFT, even though the purpose of both functions is the same. With the SURF detection we tried to detect the complete drone, but the detection had to many false detection. Another try was to detect a shape of a unique sticker, or the colored sticker we are using in the running prototype. Even if the detection was better, the false detections could not be erased, or lessened. Those were the problems we faced before deciding which method works the best for our project. Nevertheless this process is far from perfect. Many factors can hinder the program to detect the sticker and with the detection not being able to track something, the drone cannot automatically follow our drone. A first factor is the lighting of the environment. Because of the color ranges we applied in the process, if it is too dark or too bright, our method cannot detect the sticker. Another factor is the distance between the two drones. The stream has a resolution of 640 to 480 pixels. With that in mind, the sticker is small and will be represented as a few pixels within the frame. If the sticker is far away it will be represented with even less pixels, making it really hard to detect until the program loses the sticker completely. Another problem we faced is the loss of tracking of the tracked sticker on the right side of the screen. Until this day we could not find the reason why the tracking system loses the detection, if the sticker is located at the right side or going from left to right. For a correct distance measurement, the detection needs to be more consistent. Sometimes the tracking only detects a portion of the whole sticker and that results in a different distance. The last problem we faced in the autopiloting and the tracking is the Raspberry Pi. For hijacking, this small computer is enough to fulfil the process, but for image processing it has not enough processing power to parse through the images and complete the algorithm on time to coordinate the autonomous flying with the drone. For the image processing to work flawlessly on the Raspberry Pi, the algorithm would need to use less processing power. Furthermore, a different algorithm should be used to detect the whole drone.

Chapter 4

Conclusion

This project shows the issues related to the security of drones and the security threats they pose. Like we mentioned in the introduction, the issues that can arise through the use of drones, is not well known to the public. Because of this we, without any knowledge beforehand, tried to perform the hijacking of a drone and spying on a drone ourselves. We succeeded to completely hijack a drone and let it autonomously fly following a given target. Furthermore we managed to get the stream eavesdropping working, only the hardware was an issue at this part. The security threats that were mentioned could be solved in some ways. To protect drone owners, so their drones will not be stolen, the companies should set up a safer Wi-Fi for the signal transmission. Everyone that has an internet connection, has a router at home. Most of the time those routers have a Wi-Fi network to let smart phones, or similar devices, go into the internet. Those networks are protected with a security standard for Wireless networks and this is called WPA2. It is a security standard which needs a password to authorize your connection to the Wi-FI. But the drones however are not protected in any way. The WPA2 standard should be implemented in the network of the drones, which would prevent most of the hijacking attacks that can be done for the moment, because simply kicking out the user will not cut it due to not being able to connect to the drone. The other part that needs a solution is the spying or attacking with drones. To protect live presentation or speeches, festivals and similar events, they would need to have a safe zone where no drone can be flown. Jammers could be used to prevent every uncontrolled internet network to function and would hinder possible spies or attackers to launch a drone or fly it near the location. This project does not show how to perfectly hack a drone, sniff a stream or to track and autonomously fly a drone. However, we got a working prototype done, which is in line with our goals of rising the awareness for this subject, of letting people know that drones are a security threat and that security solutions should be incorporated into them.

Bibliography

[1] Melodie Bouchaud. Drones have been spotted flying over french nuclear power plants. Vice News, https://news.vice.com/article/drones-have-been-spotted-flying-over-french-nuclear-power-plants, November 2014.

[2] Agence France-Presse. More drones spotted over french nuclear power stations. The Guardian, http://www.theguardian.com/environment/2014/oct/31/more-drones-spotted-over-french-nuclear-power-stations, October 2014.

[3] Friederike Heine. Mini-drone incident shows security failings. Spiegel Online, http://www.spiegel.de/international/germany/federal-police-report-explains-inaction-despite-mini-drone-attack-a-923509.html, September 2013.

[4] Javier Iparraguirre. Installing opencv on debian testing. http://www.javieriparraguirre.net/installing-opencv-debian/, July 2014.